Modelling of Complex Systems IDP Project

senne.berden@kuleuven.be   dorde.markovic@kuleuven.be

March 2022

Due before **Sunday April 17th 23:59** on Toledo.

Note: Please first read the entire document and make sure that all parts are clear. Only then start working on your project.

1 Introduction

In this project, you will write a specification of an elevator as a dynamic system in linear time calculus (LTC). Below we give you a description of the desired behaviour of the elevator. Your task is to write a theory such that (only) the desired behaviour forms a model of the theory. To do this, you will have to work with a provided vocabulary (fixed vocabulary *V_fixed*). However, you are encouraged to introduce additional types, predicates and/or functions in a separate vocabulary (student vocabulary *V_student*), as this will make modelling much easier and will lead to a cleaner, more modular and more easily extendable specification.

Your theory will be tested by applying model expansion on several partial structures, some of which are provided to you in *structures.idp*. Some tests will check whether model expansion leads to a structure that correctly captures the intended behaviour of the elevator. Other tests will assess whether your theory correctly prohibits unintended behaviour of the elevator. Make sure that your theory gives rise to the correct behaviour when only using model expansion. In other words, the optimization of a term should not be required in order to give rise to the desired behaviour.

On the exam, you will be asked to implement extensions to the elevator system. You will have to build these extensions starting from your own base specification. This makes writing a clear, well-developed and easily-extendable specification all the more important.

Note: An additional task about invariants of the elevator system will be made available once the exercises sessions have covered invariants. This additional assignment will be a natural extension to the current project and you will be adequately notified when it is made available.

2 Description

There is a single **elevator** that serves a number of **floors**. In every time point, the elevator can **move** one floor up, one floor down, or stay on the same floor.

At any time point, one or more **stop requests** can be made. Each request is associated with a specific floor. When a request has been made, but the elevator has not yet dealt with it, the request is considered an *unanswered* request.

The elevator's scheduler responds to unanswered requests using a 'scan' approach[1]: when the elevator is responding to requests above its current floor, it continues doing so until there has been a time point in which there are no more unanswered requests at a floor higher than

[1]https://en.wikipedia.org/wiki/Elevator_algorithm

or equal to the elevator's current floor. Only then may the elevator start moving downwards to answer requests at lower floors. Similarly, when the elevator is responding to requests below its current floor, it may only start moving upwards again when there has been a time point in which there are no more unanswered requests at floors lower than or equal to the elevator's current floor. In case there are no unanswered requests at time point t, but at time point $t + 1$ there are several new unanswered requests both above and below the elevator's current floor, the elevator will start going in the direction of the nearest request. In case of a tie, the elevator will prioritize the upwards direction.

The elevator has **doors**, which take a single time step to open or close. Once an elevator has stopped at a floor to answer an unanswered request, its doors will take one time step to open and then take another time step to close again; only once the doors are closed again is the request considered *answered*. The elevator can then continue to respond to other requests, if there are any. The elevator only moves to respond to requests. When the last unanswered request has been answered, the elevator stays still with closed doors until a new request is made.

Note that the elevator *has to* answer requests in the appropriate way. For example, it may not stay idle when there are remaining unanswered requests. The correct behavior of the elevator follows deterministically from the made requests. Consequently, performing model expansion on the partial structures provided in *structures.idp* should result in exactly one model.

2.1 Formalization

2.1.1 Vocabulary

The supplied fixed LTC vocabulary is the following:

```
LTCvocabulary V {
    type Time isa nat
    Start: Time
    partial Next(Time): Time

    type Floor isa nat
    type DoorState constructed from {Closed, Open}

    // Fluents
    ElPosition(Time): Floor
    ElDoorState(Time): DoorState
    UnansweredRequest(Time, Floor)

    // Actions
    MakeRequest(Time, Floor)
}
```

with the following intended meaning:

- **Time** is a set of time points.
- **Start** is the initial time point.
- **Next(t)** is the successor time point of time point t.
- **Floor** is the set of floors.
- **DoorState** is a set of possible states of the elevator's door, consisting of *Closed* and *Open*.
- **ElPosition(t)** denotes the floor the elevator is on at time point t.

- **ElDoorState(t)** denotes the state of the elevator's doors at time point *t*.
- **UnansweredRequest(t, f)** denotes that there is an unanswered request at floor *f* at time point *t*.
- **MakeRequest(t, f)** denotes that a request is made for floor *f* at time point *t*.

2.1.2 Time

Time is linear and is interpreted by natural numbers (LTC - Linear Time Calculus).

2.1.3 Floors

The elevator can go to a number of floors. A floor is characterized by a natural number. The total number of floors can differ from structure to structure, but the floor numbers always start from 0, i.e., the zeroth floor is always the bottom floor.

2.1.4 The elevator

There is a single elevator, with two dynamic properties: its door state and its position.

The door state denotes whether the elevator's doors are open or closed. The elevator's door state should only change in order to answer a request. Changing the elevator's door state should take one time step. In the first time step, the elevator's doors should be closed.

The position denotes the floor the elevator is at. In every time step, the elevator either moves one floor up, moves one floor down, or remains at the same floor. The elevator is only allowed to move when its doors are closed. The elevator is only allowed to move up when it is not on the top floor, and is only allowed to move down when it is not on the bottom floor. In the first time step, the elevator should be at the bottom floor.

In any time step, the elevator can alter at most one property. In other words, it cannot change its position and its door state in a single time step.

2.1.5 Requests

Requests which the elevator still has to answer are represented with predicate *UnansweredRequest.* A request is considered answered when the elevator has gone to the corresponding floor, opened its doors and then closed its doors again. In other words, if the elevator has gone to floor f to answer an unanswered request, and has its doors open (on floor f) at time point t and closed again at time point $t + 1$, then the request is considered answered starting from time point $t + 1$.

Requests will be made using the *MakeRequest* action. The value of this predicate will be part of the partial structures given to you (see *structures.idp*). When *MakeRequest(t, f)* holds, and *UnansweredRequest(t, f)* does not hold, then *UnansweredRequest* for floor f should hold starting from time step $t + 1$, until the request has been answered. *MakeRequest(t, f)* should only take effect if *UnansweredRequest(t, f)* does not hold.

2.2 Visualization

In figure 1, an example of a visualization generated by IDP is shown. In the top-left corner, the current time step is displayed. There are six floors. The elevator is currently at floor 0 with closed doors, hence it is shown in red. When the elevator's doors are open, it is shown in green. There is currently one unanswered request, on floor 4. By clicking on the visualization, either to the left or to the right of the elevator shaft, you can progress time by one time step.

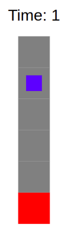

Figure 1: An example visualization

2.3 Illustration of intended behaviour

Consider the following partial structure

```
Structure S:V_student {
    Time = {0..19}
    Floor = {0..5}
    MakeRequest = {0, 5; 2, 3; 3, 2; 15, 3}
}
```

For this partial structure, the intended behaviour of the elevator is shown in figure 2. Note the following:

- In the beginning, the elevator goes up to handle the request on the 5th floor, but on the way there, another request pops up on the 3rd floor. The elevator should stop to handle this call. In case the request would appear exactly at the current position of the elevator, the elevator should also stop to handle it first.
- At time point 4 a new request is made at the 2nd floor, which is closer to the elevator than the one on the 5th floor. Still, the elevator should not change its direction; it should continue going up until all requests above are handled.

3 Tips and tricks

- The IDP IDE supports autocompletion (CTRL+SPACE); use this to save yourself some time.
- It might be helpful to first focus on modelling the elevator's movement and make sure it visits floors in the appropriate order, and to only then focus on the behaviour of the elevator's doors.
- It is often helpful to introduce new types, predicates and/or functions in the student vocabulary, and to then define and use them in the student theory. For example, consider introducing action predicates that can be used to alter the elevator's position and door state. Introducing new symbols can make your theory more clear, more modular and easier to write and extend.
- Some structures are provided in *structures.idp*. Use these to check the behaviour arising from your specification. Do note that these provided structures are not meant to be exhaustive: if your specification leads to the correct behaviour on these structures, this does not guarantee that your specification leads to the correct behaviour on all structures. In addition to the provided structures, your specification will be evaluated on several other structures. For this reason, you are encouraged to write additional structures to test the correctness of your specification with.

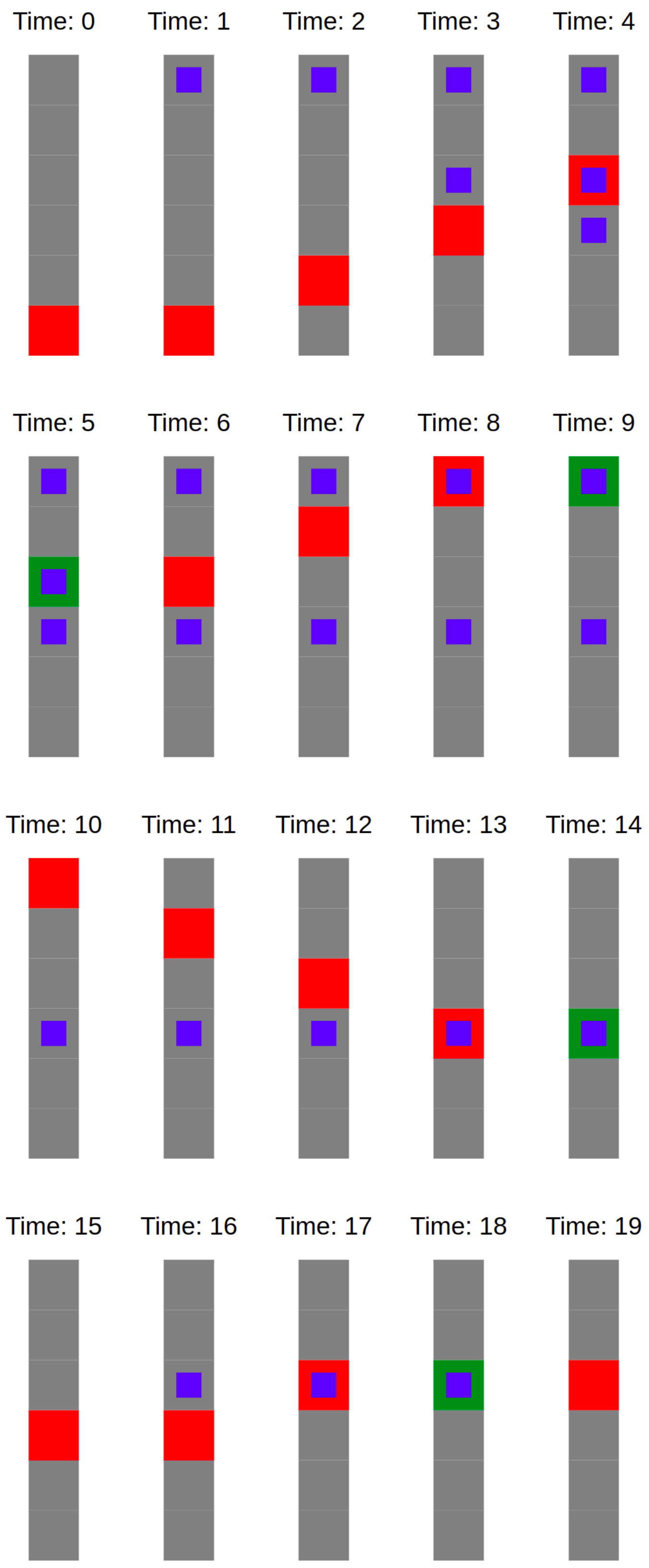

Figure 2: An illustration of the intended elevator behaviour

4 Practical

4.1 Provided files

On Toledo you can find a `skeleton.zip` file containing:

1. IDP skeleton file `elevator.idp` consisting of:
    - A fixed vocabulary *V_fixed* (**You cannot add anything to this**)
    - A student vocabulary *V_student* extending the fixed vocabulary (This is the place to add extra types, predicate and/or function symbols)
    - A fixed theory *T_time* (**You cannot add anything to this**)
    - An empty student theory over the student vocabulary (You need to add sentences to this)
    - The `main` procedure that searches for one model and visualizes it. In order to try different scenarios you may alter the structure provided to method *onemodel*. Changing other parts of the main procedure is allowed for debugging purposes, but make sure to correctly restore it once you are done.
2. `structures.idp` contains various structures over *V_student* vocabulary. Each structure specifies a sequence of *make request* actions. You should use these to test your elevator formalization.
3. `visualize_elevator.idp` file and `idpd3` directory are needed for the visualization and you may not edit or move them.

4.2 Project output

The output of this project must consist of a `.zip` file with file name `Lastname_Firstname_StudentNumber` (where Lastname and Firstname are your actual names, and StudentNumber is your student number, starting with *r* or *s*) containing the following:

1. `elevator.idp`
2. A *concise* report (at most one page) named `report.txt` in which you discuss design decisions.
    - For each new predicate symbol `p` and function symbol `f` you introduce, the report should contain its intended interpretation in the following format:
        - `Pred(x,y)` is true if and only if ⟨some condition on x and y here⟩,
        - `Func(x,y,z)` is ⟨some description here⟩.
    - The report should also contain an **estimation of the time** you spent on this part of the project in hours.

Note: Do not compress the directory that contains required files but **only files** themself. Do not add anything else except the required two files!

4.3 About the specification

- You have to start from the provided `.idp` files.
- You are **not allowed** to change the fixed vocabulary or the fixed theory at all (not even renaming).
- You are allowed (and advised) to add new symbols to the student vocabulary. Examples are action predicates and cause predicates for fluents. You are also allowed to add extra types.
- Use well-chosen names for introduced symbols and variables.
- Make your specification clear and readable.
- Use a consistent form/layout in your specification.
- Use consistent indentation.
- Avoid overly long sentences.
- Write comments: clearly specify the meaning of every line in your theory.
- When quantifying a variable, always specify the variable's type.
- Make sure there are no warnings and errors, except for the *Verifying and/or autocompleting structure* warnings and the errors on including files.

4.4 Additional task

As mentioned in the introduction, there will be one additional task regarding this project. This task will be related to proving invariants of a dynamic system. You will receive additional materials and details via Toledo. You will also receive a reminder email.

4.5 Grading

This part of the project will primarily be graded using automatic tests. However, **the majority of the grades for this project will be handed out on the exam**, where we will ask you to implement extensions to the elevator system. You will build these extensions starting from your own specification. This makes writing a clear, well-developed and easily-extendable specification all the more important.

4.6 Restrictions

You are allowed to discuss this project with others, but are not allowed to copy any code from other students. This is not an open source project, i.e., you are not allowed to put your code openly available on the web. If you want to use Git, do not use public GitHub repositories.

4.7 Discussion board

You can use the discussion forum "Questions about IDP project" (Toledo → MCS course → discussion board) to post your questions and to discuss issues with your colleagues and teaching assistants. In this way, you can find answers to your questions faster and save some time. You are allowed to include IDP code in your questions if desired (e.g., to illustrate the problem you are facing), on condition that this code is **not related to this project**. In other words, if you post IDP code on the discussion forum, this code **may not be related to the elevator system**. If you are not sure whether your code is allowed to be added to a question on the discussion board, first send your code and question to the teaching assistants via e-mail.

Good luck!
In case you have any questions, do not hesitate to contact the teaching assistants.